UNIT – 4 : Introduction To Flutter

1. What is Flutter?

Flutter is an Open Source Cross Platform Development Framework which use Dart language presented on 2015 at Dart Developer Summit.

Flutter is created by Google and Launched in 2016 and begin to get adopted by large community, and first stable version was released at the end of 2018.

Flutter build Native Mobile App for both Android and iOS with single code base. There are many other Frameworks also available with same goals like

Flutter is an open-source UI software development kit created by Google. It is used to develop cross platform applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia, Web platform, and the web from a single codebase.

- React Native
- PhoneGap
- Xamarine
- Cordova

2. Explain Features of Flutter?

Flutter has some features which makes similar to native app development like:

- High performance
- Full control of the user interface
- Dart language
- Being backed by Google
- Open source framework
- Developer resources and tooling

1. High Performance:

Flutter is developed to renders widgets with a high frame rate. Existing frameworks is based on JavaScript rendering, which causes overheads in performance because everything is drawn using Web View. Other framework uses Original Equipment Manufacturer (OEM) widgets but with a bridge to request the API to render the components. This needs an extra step to render the user interface (UI).

2. Full control of the user interface:

The Flutter framework chooses to do all the UI by itself, rendering the visual components directly to the canvas.

3. Dart Language:

Dart is a general-purpose, high-level programming language developed and released by Google in 2017. Dart language gain its popularity after flutter was released. Its syntax is same as Java and C and JavaScript language. Dart is an open-Source Programming language specially designed for Mobile App Development but can also used for following types of development.

- Web Development
- Desktop Development
- Cross Platform Mobile Development
- IOT

It supports modern programming techniques like interfaces, generic types, type interface, classes and objects, etc.

It supports two types of compilation.

1. JIT (Just In Time)
2. AOT(Ahead of Time)

JIT converts Dart code to JavaScript code using dart2js tool to run dart code on all modern web browser.

AOT converts bytecode to machine code.

Features of Dart

1. It is Object Oriented Language and supports all features of OOP.
2. It is Open-source comes with BSD licence, it is free and available publicly.
3. It is Platform independent like java and can run on any platform using DVM (Dart Virtual Machine)
4. Dart is Stable and very useful for real time applications.
5. It is easy to learn as it is similar to Java, JavaScript and C language.
6. Dart performs both static type check and run time type checking, as it also supports dynamic type.

3. Write a note on Architecture of Flutter:

In this section, we are going to discuss the architecture of the Flutter framework. The Flutter architecture mainly comprises of four components.

1. Flutter Engine
2. Foundation Library
3. Widgets
4. Design Specific Widgets

Flutter Engine

It is a portable runtime for high-quality mobile apps and primarily based on the C++ language. It implements Flutter core libraries that include animation and graphics, file and network I/O, plugin architecture, accessibility support, and a dart runtime for developing, compiling, and running Flutter applications. It takes Google's open-source graphics library, **Skia**, to render low-level graphics.

Foundation Library

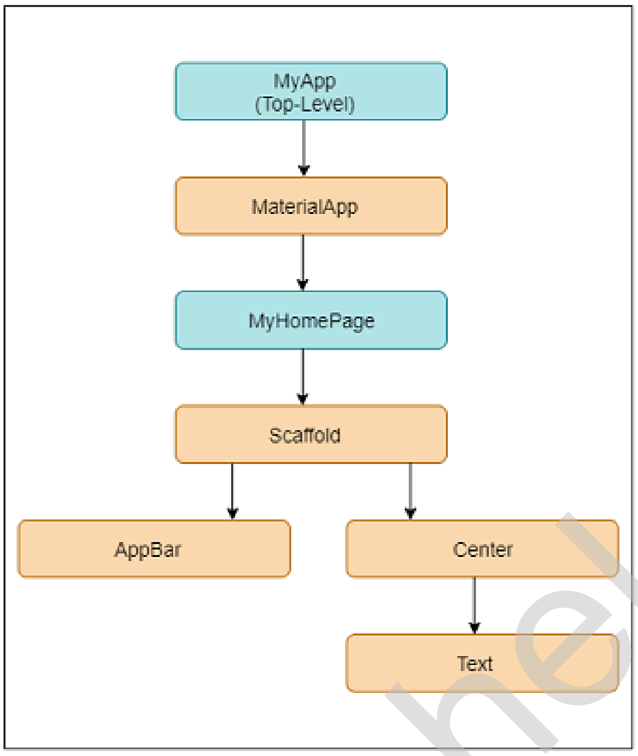
It contains all the required packages for the basic building blocks of writing a Flutter application. These libraries are written in Dart language.

Widgets

In Flutter, everything is a widget, which is the core concept of this framework. Widget in the Flutter is basically a user interface component that affects and controls the view and interface of the app. It represents an immutable description of part of the user interface and includes graphics, text, shapes, and animations that are created using widgets. The widgets are similar to the React components.

In Flutter, the application is itself a widget that contains many sub widgets. It means the app is the top-level widget, and its UI is build using one or more children widgets, which again includes sub child widgets. This feature helps you to create a complex user interface very easily.

We can understand it from the hello world example created in the previous section. Here, we are going to explain the example with the following diagram.



In the above example, we can see that all the components are widgets that contain child widgets. Thus, the Flutter application is itself a widget.

Design Specific Widgets

The Flutter framework has two sets of widgets that conform to specific design languages. These are Material Design for Android application and Cupertino Style for IOS application.

Gestures

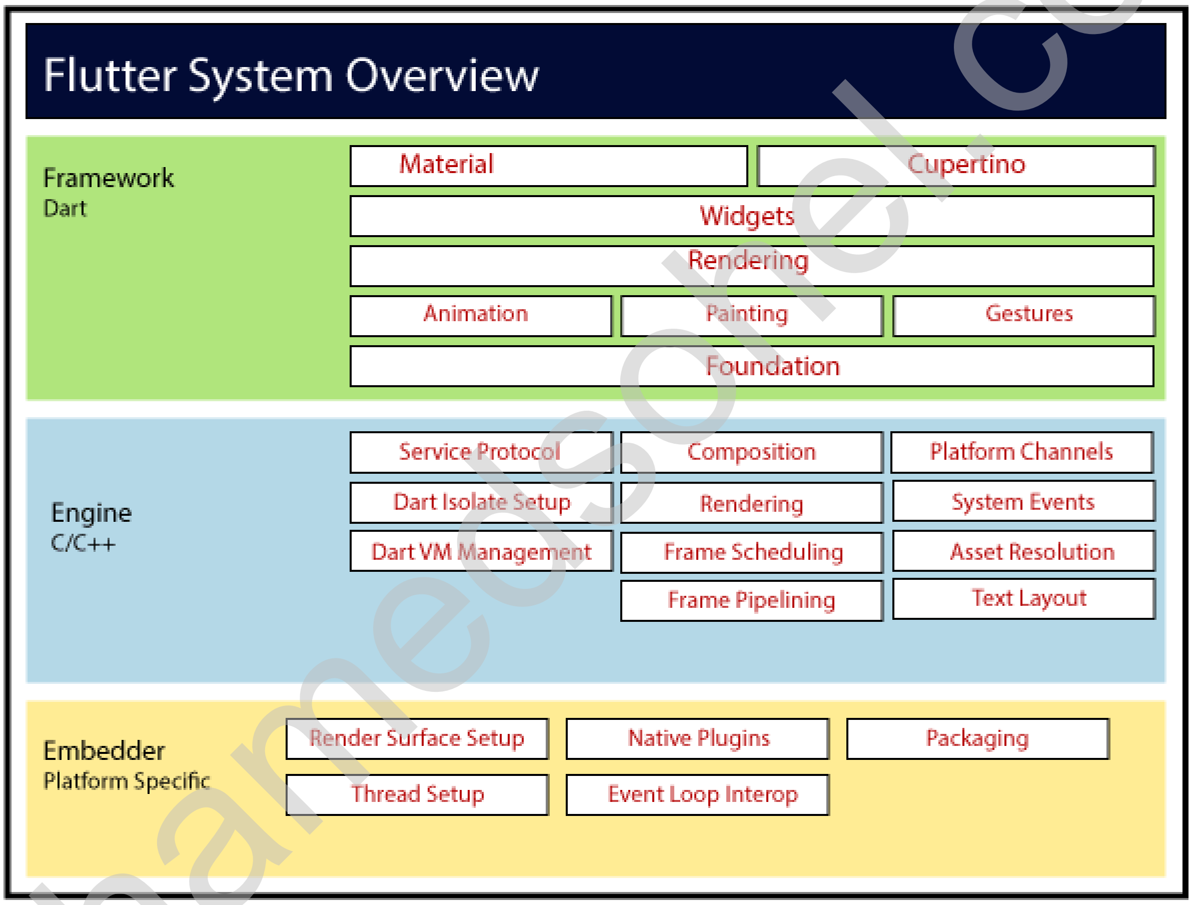
It is a widget that provides interaction (how to listen for and respond to) in Flutter using GestureDetector. **GestureDector** is an invisible widget, which includes tapping, dragging, and scaling interaction of its child widget. We can also use other interactive features into the existing widgets by composing with the GestureDetector widget.

State Management

Flutter widget maintains its state by using a special widget, StatefulWidget. It is always auto re-rendered whenever its internal state is changed. The re-rendering is optimized by calculating the distance between old and new widget UI and render only necessary things that are changes.

Layers

Layers are an important concept of the Flutter framework, which are grouped into multiple categories in terms of complexity and arranged in the top-down approach. The topmost layer is the UI of the application, which is specific to the Android and iOS platforms. The second topmost layer contains all the Flutter native widgets. The next layer is the rendering layer, which renders everything in the Flutter app. Then, the layers go down to Gestures, foundation library, engine, and finally, core platform-specific code. The following diagram specifies the layers in Flutter app development.



4. Explain Flutter Widgets in detail:

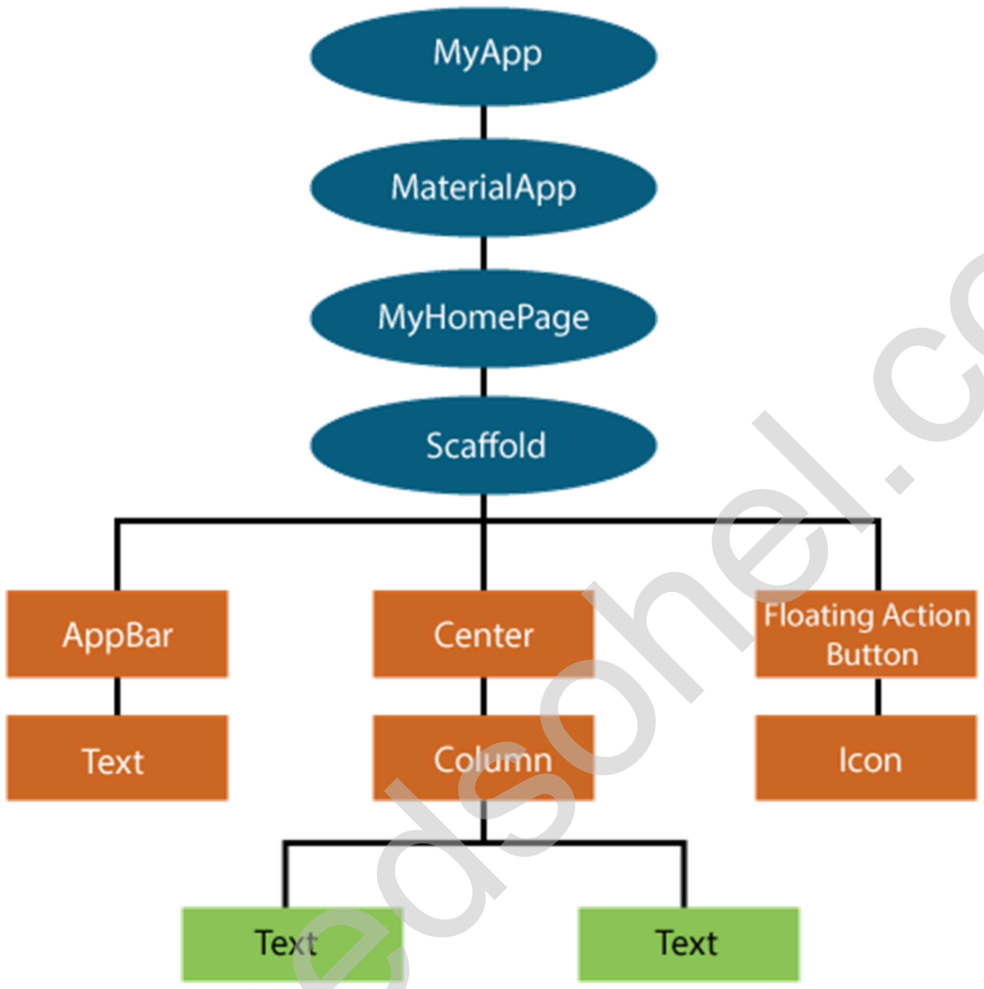
In this article we will discuss what is Widgets, Types of Widgets and How we can use it in Flutter App.

Flutter Apps are created with Widgets. Flutter has widgets for everything or we can say everything created in flutter is by widgets only.

Widgets are dart classes which can be used to create effective UI, Layout, Animation, Services etc.

Whenever we create anything in flutter it is inside widget. Flutter App Screen follows Widget tree like structure. Widgets are nested with each other to build the app screen.

The root of app is a widget itself and it ends with a widget also.



Widgets are divided into three categories based on its types.

1. Visible Invisible Widgets
2. Stateful and Stateless Widgets
3. Single Child and Multi-child Widgets

Visible and Invisible Widgets

The Widgets which we can see on screen are visible widgets, they are useful to show controls or elements on the screen like Text, Image, Input Field, Button etc.

Invisible Widgets are those which do not appear on screen but useful to provide layouts and services to the app screen. These widgets are also called Layout Widgets. They are responsible to handle screen layout in which visible widgets can be arranged. Scaffold, Column, Row, Container are some examples of Invisible Widgets.

Stateful and Stateless Widget

User Interface are never static, it always changes based on user interactions. In flutter Widgets are immutable but they are not final, they need to change during the life cycle of the App. Because of this reason Flutter provides two types of Widgets. Stateful and Stateless.

While creating UI, developers are responsible to choose the kind of widget. The biggest difference between stateful and stateless is the way the build.

Stateless Widget: Application contains many widgets that will never change their properties after instantiated. These widgets do not have any state associated with them, and they do not change themselves when some action or behaviour occurs on screen. These widgets are created within stateless widget class and control by its parent widget. Means the child widget will receive description from parent control and will not changed anytime by itself. stateless widgets have only final properties defined during construction.

Stateful Widget : Stateful Widget change their description dynamically during their life cycle. These widgets are also immutable but they are associated with State class which represent their current state. Framework will rebuild the widget whenever the state has been changed by any part of the application.

The State object will notify whenever the widget needs to rebuild that cause the update in element tree.

Single Child and Multi-Child Widgets

Flutter Widgets can hold only one child in it. A widget with child property can have only one child. For example, Center Widget, which is a layout widget can have only one child to show at center of the screen.

Flutter also has some layout widgets which can have multiple children or an array of child. Example, Row and Column, In this widget we can have array of Widgets which build on screen row wise or Columns wise, depending on the Layout.

5. Explain Visible Widget of Flutter:

1. Visible widget:

The visible widgets are related to the user input and output data. Some of the important types of this widget are:

a.) Text
b.) Button
c.) Image
d.) icon

a.) Text:

A Text widget holds some text to display on the screen. We can align the text widget by using textAlign property, and style property allow the customization of Text that includes font, font weight, font style, letter spacing, color, and many more. We can use it as like below code snippets.

```
new Text(
'Hello, Javatpoint!',
textAlign: TextAlign.center,
style: new TextStyle(fontWeight: FontWeight.bold),
)
```

b.) Button:

This widget allows you to perform some action on click. Flutter does not allow you to use the Button widget directly; instead, it uses a type of buttons like a FlatButton and a RaisedButton. We can use it as like below code snippets.

```
//FlatButton Example
new FlatButton(
child: Text("Click here"),
onPressed: () {
// Do something here
},
),

//RaisedButton Example
new RaisedButton(
child: Text("Click here"),
elevation: 5.0,
onPressed: () {
// Do something here
},
),
```

In the above example, the onPressed property allows us to perform an action when you click the button, and elevation property is used to change how much it stands out.

c.) Image:

This widget holds the image which can fetch it from multiple sources like from the asset folder or directly from the URL. It provides many constructors for loading image, which are given below:

- Image: It is a generic image loader, which is used by ImageProvider.
- asset: It load image from your project asset folder.
- file: It loads images from the system folder.
- memory: It load image from memory.
- network: It loads images from the network.

To add an image in the project, you need first to create an assets folder where you keep your images and then add the below line in pubspec.yaml file.

```
assets:
  - assets/
```

Now, add the following line in the dart file.

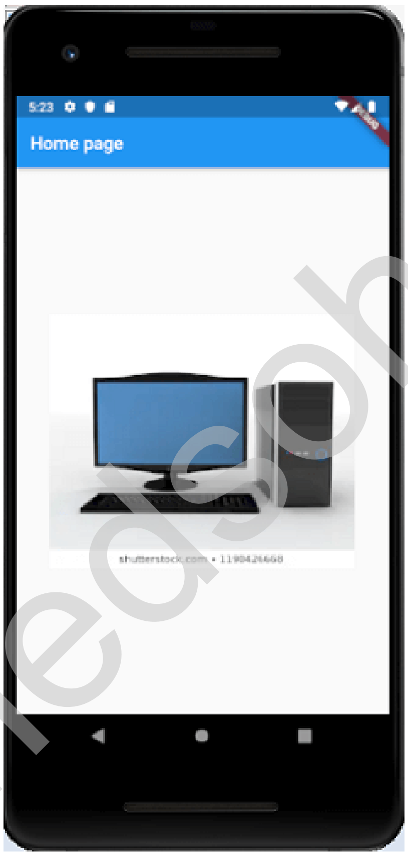
```
Image.asset('assets/computer.png')
```

The complete source code for adding an image is shown below in the hello world example.

```
class MyHomePage extends StatelessWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  // This widget is the home page of your application.
  final String title;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(this.title),
      ),
      body: Center(
        child: Image.asset('assets/computer.png'),
      ),
    );
  }
}
```

When you run the app, it will give the following output.



d.) Icon:

This widget acts as a container for storing the Icon in the Flutter. The following code explains it more clearly.

```
new Icon(
  Icons.add,
  size: 34.0,
)
```

2. Invisible widget:

The invisible widgets are related to the layout and control of widgets. It provides controlling how the widgets actually behave and how they will look onto the screen. Some of the important types of these widgets are:

Column

A column widget is a type of widget that arranges all its children's widgets in a vertical alignment. It provides spacing between the widgets by using the mainAxisAlignment and crossAxisAlignment properties. In these properties, the main axis is the vertical axis, and the cross axis is the horizontal axis.

Example

The below code snippets construct two widget elements vertically.

```
new Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    new Text(
      "VegElement",
    ),
    new Text(
      "Non-vegElement"
    ),
  ],
),
```

Row

The row widget is similar to the column widget, but it constructs a widget horizontally rather than vertically. Here, the main axis is the horizontal axis, and the cross axis is the vertical axis.

Example

The below code snippets construct two widget elements horizontally.

```
new Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: <Widget>[
    new Text(
      "VegElement",
    ),
    new Text(
      "Non-vegElement"
    ),
  ],
),
```

Center

This widget is used to center the child widget, which comes inside it. All the previous examples contain inside the center widget.

Example

```
Center(
  child: new clumn(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: <Widget>[
      new Text(
        "VegElement",
      ),
      new Text(
        "Non-vegElement"
      ),
    ],
  ),
),
```

Padding

This widget wraps other widgets to give them padding in specified directions. You can also provide padding in all directions. We can understand it from the below example that gives the text widget padding of 6.0 in all directions.

Example

```
Padding(
  padding: const EdgeInsets.all(6.0),
  child: new Text(
    "Element 1",
  ),
),
```

Scaffold

This widget provides a framework that allows you to add common material design elements like AppBar, Floating Action Buttons, Drawers, etc.

Stack

It is an essential widget, which is mainly used for overlapping a widget, such as a button on a background gradient.